# Mobile Application Development

Produced by

David Drohan (ddrohan@wit.ie)

Department of Computing & Mathematics
Waterford Institute of Technology
http://www.wit.ie

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

# Android Anatomy

# Agenda

- Overview of Android Application Components
- The Android Application Life Cycle
- The Online Developer Resources
- The "*Donation*" Case Study – a first (very brief!) look…

# Android App Components

- App components are the essential building blocks of an Android app. Each component is a different point through which the system can enter your app.
- Not all components are actual entry points for the user and some depend on each other, but each one exists as its own entity and plays a specific role—each one is a unique building block that helps define your app's overall behavior.
- There are four different types of app components. Each type serves a distinct purpose and has a distinct lifecycle that defines how the component is created and destroyed.
- We'll briefly mention a few other components (of sorts) that also make up your App.

# Android App Components

## 1. Activities

- represents a single screen with a user interface
- acts as the ‘controller’ for everything the user sees on its associated screen
- implemented as a subclass of Activity
- e.g. email app (listing your emails)

## 2. Services

- a component that runs in the background to perform long-running operations or to perform work for remote processes
- does not provide a user interface
- can be started by an activity
- is implemented as a subclass of Service
- e.g. music player (playing in background)

# Android App Components

## 3. Content Providers

- manages a shared set of app data
- can store the data in the file system, an SQLite database, on the web, or any other persistent storage location your app can access
- through the content provider, other apps can query or even modify the data
- e.g. Users Contacts (your app could update contact details)

## 4. Broadcast Receivers

- a component that responds to system-wide broadcast announcements
- broadcasts can be from both the system and your app
- implemented as a subclass of BroadcastReceiver and each broadcast is delivered as an Intent object
- e.g. battery low (system) or new email (app via notification)

# How it all Fits Together *

**Big N.B.**
**for all these!**

- Based on the *Model View Controller* design pattern.
- Don't think of your program as a linear execution model:
  - Think of your program as existing in logical blocks, each of which performs some actions.
- The blocks communicate back and forth via message passing (*Intents*)
  - Added advantage, **physical user interaction (screen clicks) and inter process interaction can have the same programming interface**
  - Also the OS can bring different pieces of the app to life depending on memory needs and program use

See the Appendix for a more detailed explanation of these components

- For each distinct logical piece of program behavior you'll write a Java class (derived from a base class).
- Activities/Fragments: Things the user can *see* on the screen. Basically, the 'controller' for each different screen in your program.
- Services: Code that isn't associated with a screen (background stuff, fairly common)
- Content providers: Provides an interface to exchange data between programs (usually SQL based)
- You'll also design your layouts (screens), with various types of widgets (Views), which is what the user sees via *Activities & Fragments*

# The (Application) Activity Life Cycle *

- Android is designed around the unique requirements of mobile applications.
  - In particular, Android recognizes that resources (memory and battery, for example) are limited on most mobile devices, and provides mechanisms to conserve those resources.
- The mechanisms are evident in the *Android Activity Lifecycle*, which defines the states or events that an activity goes through from the time it is created until it finishes running.

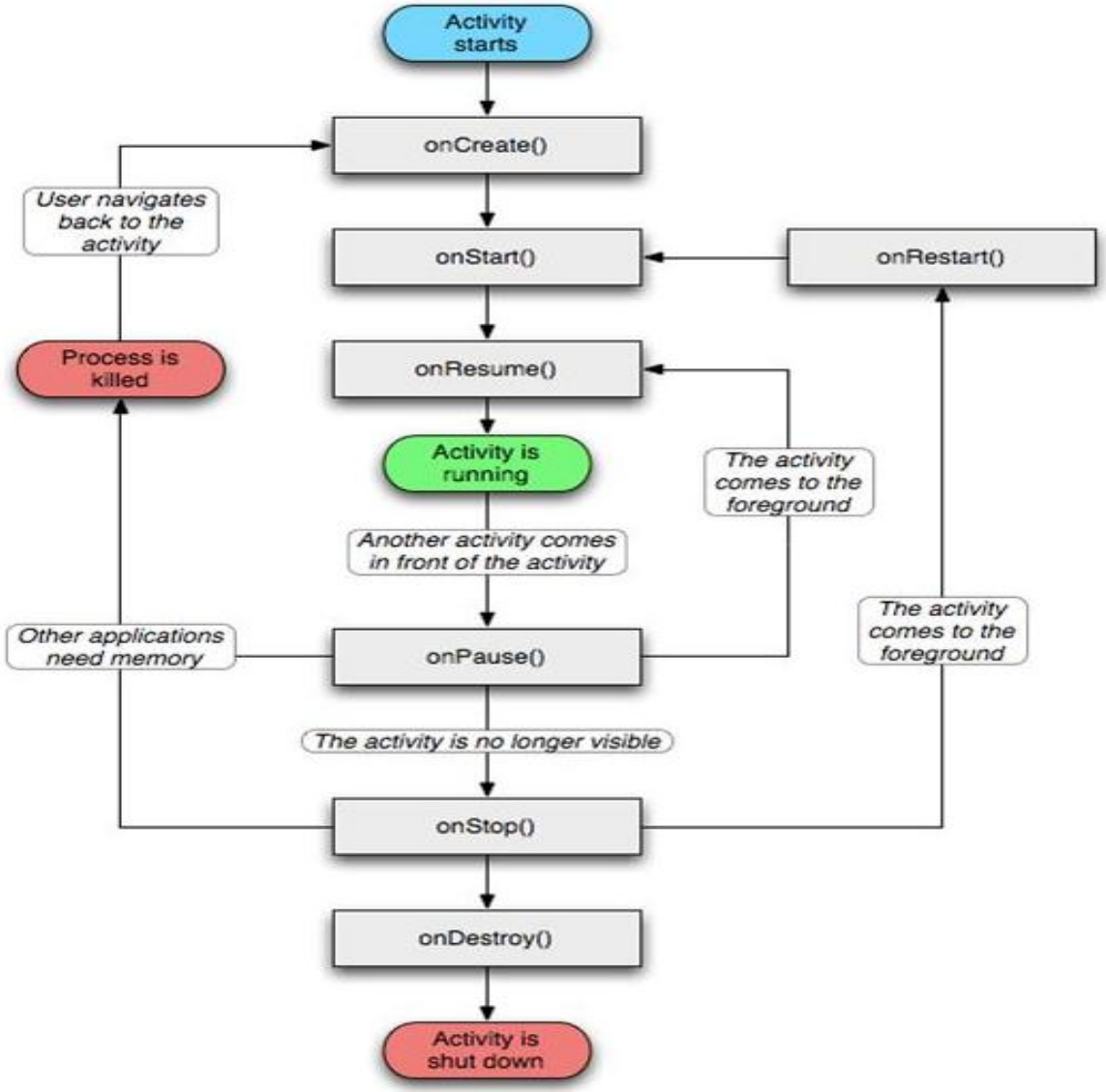
See the Appendix for a more detailed explanation of these ‘states’

# The (Application) Activity Life Cycle

- An application itself is a set of activities with a Linux process to contain them
  - However, an application DOES NOT EQUAL a process
  - Due to (the previously mentioned) low memory conditions, an activity might be suspended at any time and its process be discarded
    - The activity manager remembers the state of the activity however and can reactivate it at any time
    - Thus, an activity may span multiple processes over the life time of an application

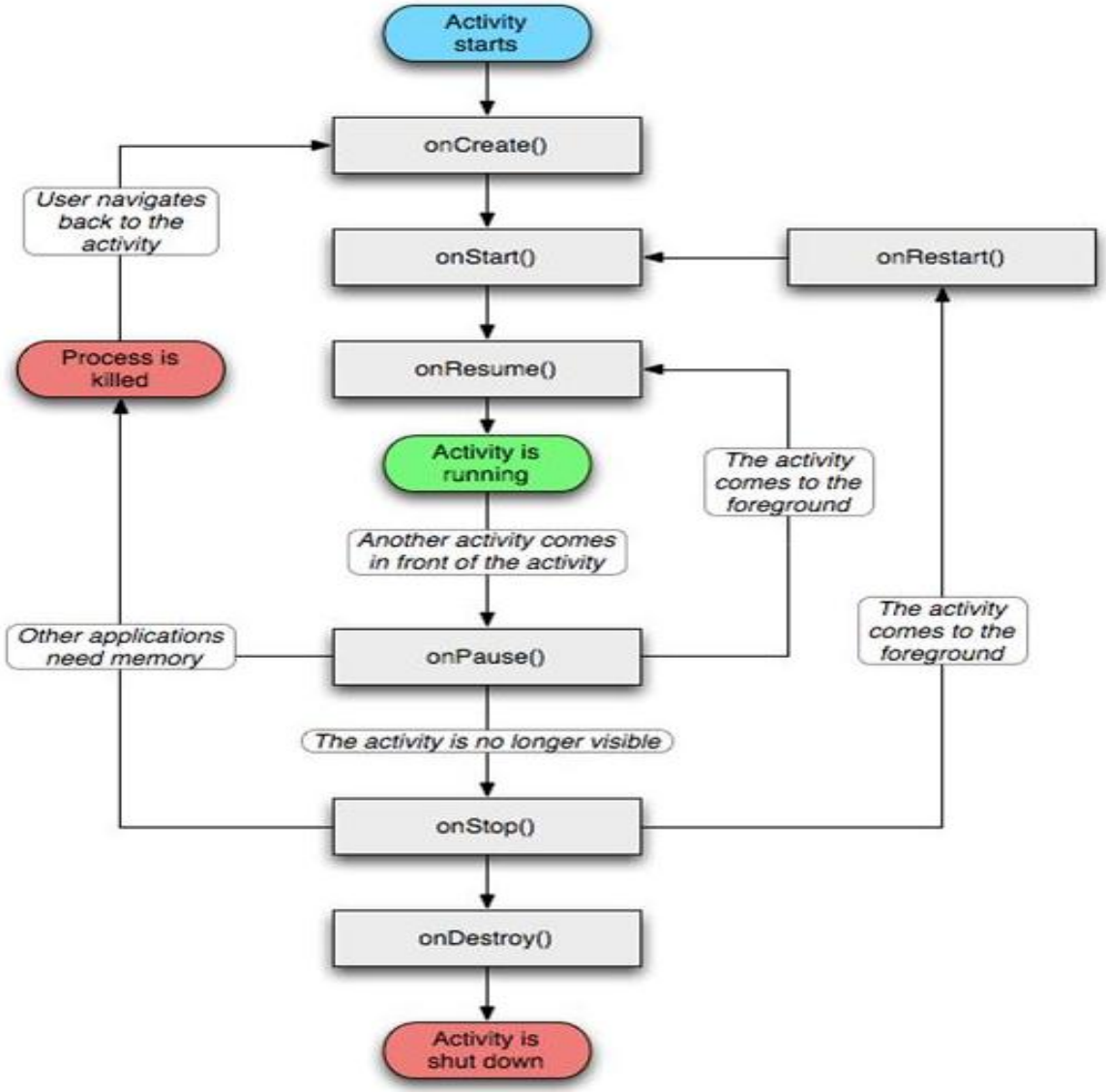
# The Activity Life Cycle *

- The Activity has a number of predefined functions that you override to handle events from the system.
- If you don't specify what should be done the system will perform the default actions to handle events.
- Why would you want to handle events such as **`onPause()`**, etc… ?
  - You will probably want to do things like release resources, stop network connections, back up data, etc…



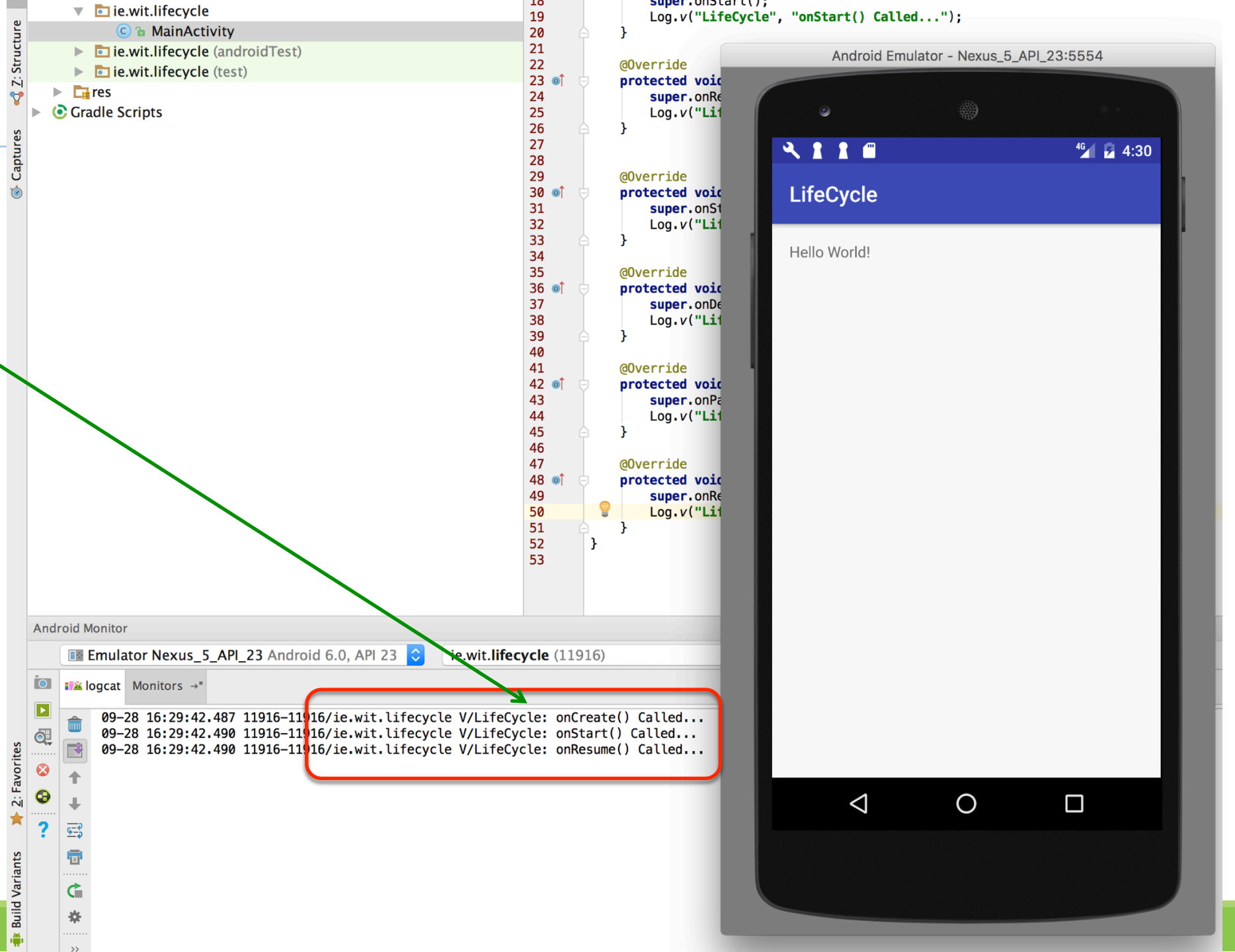
# The Activity Life Cycle

- At the *very minimum* ,you need (and is supplied) **`onCreate()`**
- **`onStop()`** and **`onDestroy()`** are optional and may never be called
- If you need persistence, the save needs to happen in **`onPause()`**



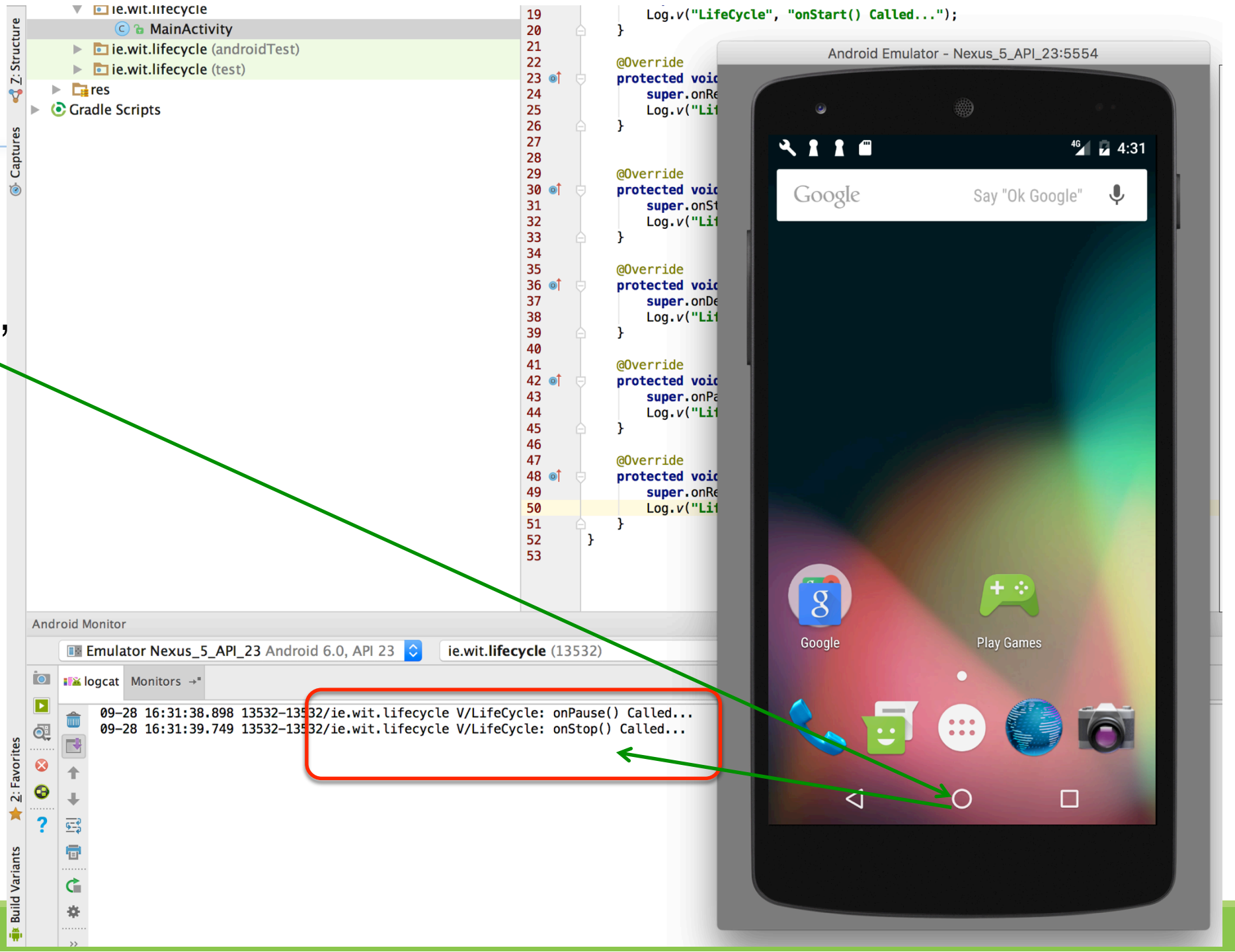
# LifeCycle Example (1) *

User Launches App



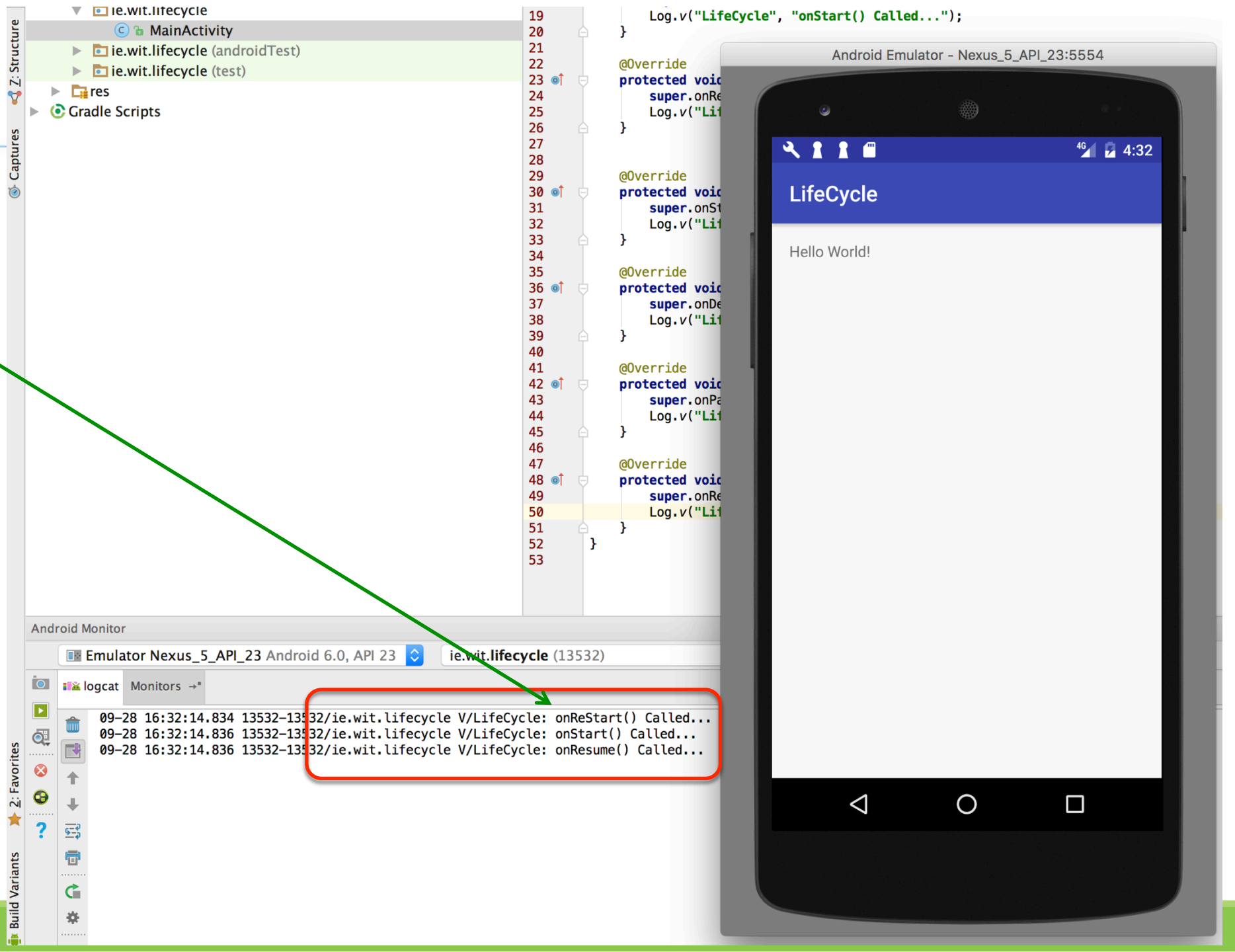
# LifeCycle Example (2) *

User Selects ‘Home’



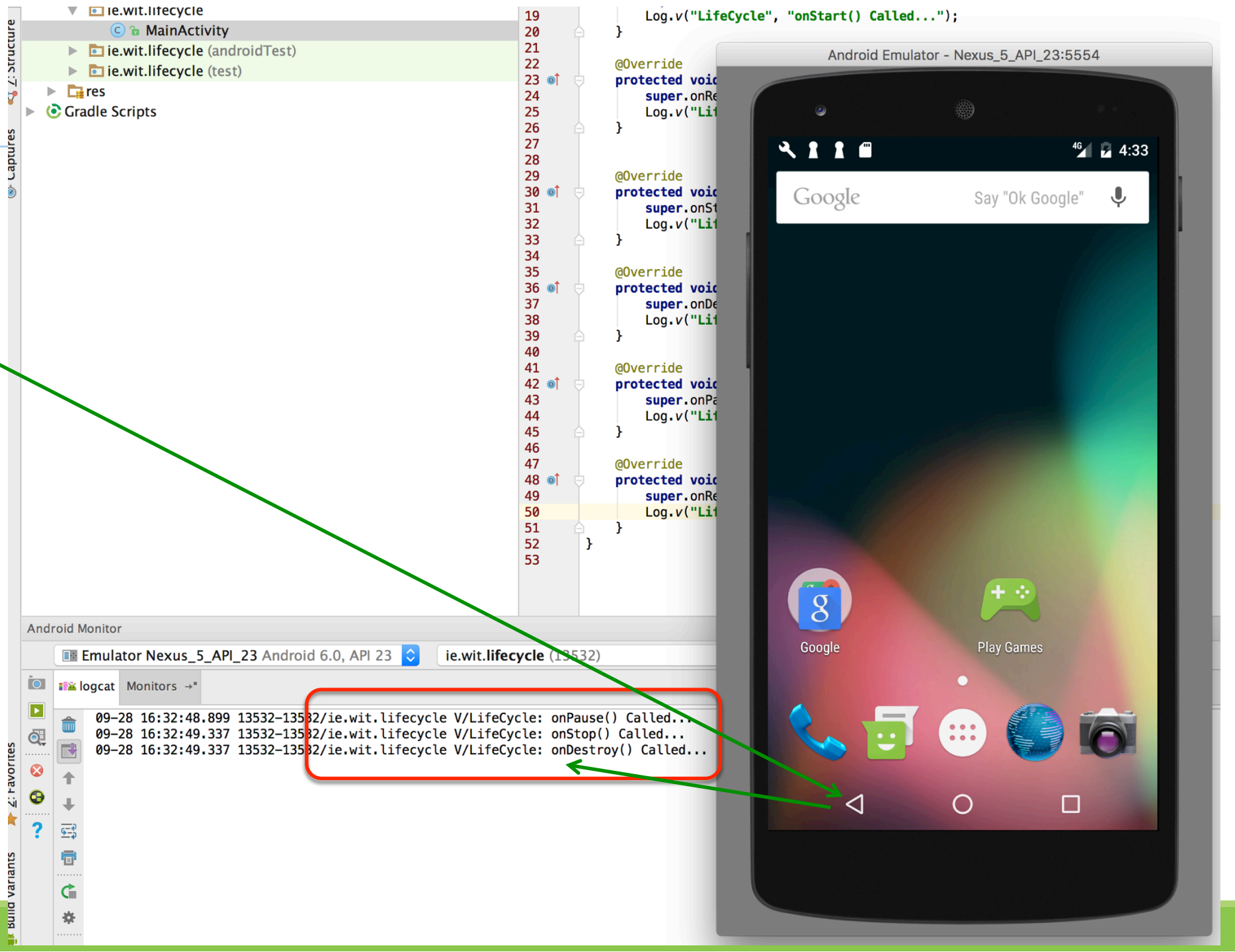
# LifeCycle Example (3) *

User restarts App



# LifeCycle Example (4) *

User Selects ‘Back’



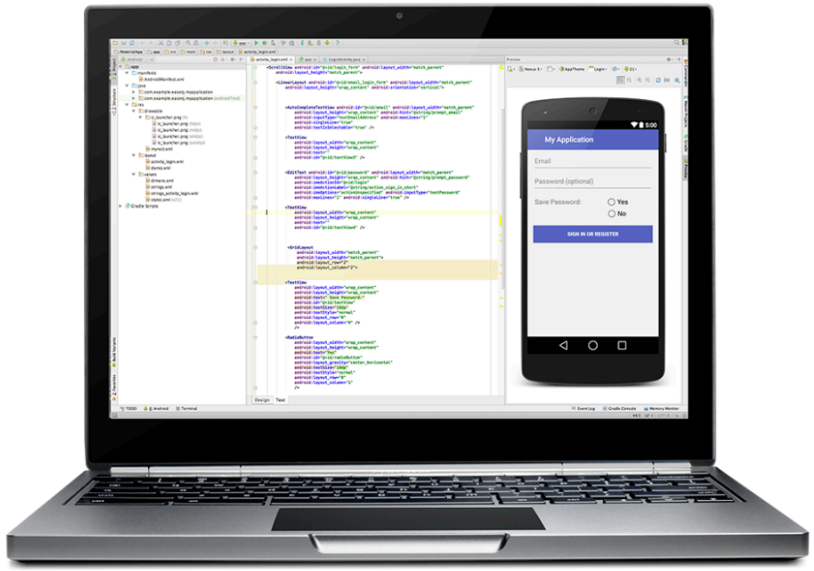
# So, after all that, how do I Design my App?

- The way the system architecture is set up is fairly open:
  - App design is somewhat up to you, but you still have to live with the Android execution model.
- Start with the different *screens/ layouts* (Views) that the user will see. These are controlled by the different *Activities* (Controllers) that will comprise your system.
- Think about the *transitions* between the screens, these will be the *Intents* passed between the Activities.
- Think about what background *services* you might need to incorporate.
  - Exchanging data
  - Listening for connections?
  - Periodically downloading network information from a server?
- Think about what *information* must be stored in long term memory (SQLite) and possibly design a content provider around it.
- Now connect the Activities, services, etc… with Intents…
- Don't forget good OOP ☺ and
- USE THE DEVELOPER DOCs & GUIDES (next few slides)

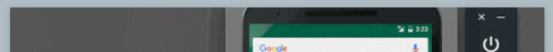
# Get Started with Android Studio

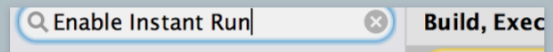
Everything you need to build incredible app experiences on phones and tablets, Wear, TV, and Auto.

- Set up Android Studio
- Build your first app
- Learn about Android
- Sample projects



## Latest





# Getting Started

Welcome to Training for Android developers. Here you'll find sets of lessons within classes that describe how to accomplish a specific task with code samples you can re-use in your app. Classes are organized into several groups you can see at the top-level of the left navigation.

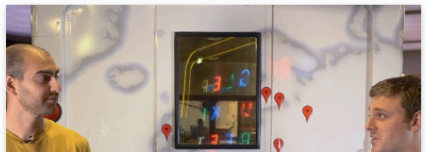
This first group, *Getting Started*, teaches you the bare essentials for Android app development. If you're a new Android app developer, you should complete each of these classes in order.

If you prefer to learn through interactive video training, check out this trailer for a course about the fundamentals of Android development.

START THE VIDEO COURSE


## Online video courses

If you prefer to learn through interactive video training, check out these free courses.



# Introduction to Android

Android provides a rich application framework that allows you to build innovative apps and games for mobile devices in a Java language environment. The documents listed in the left navigation provide details about how to build apps using Android's various APIs.

To learn how apps work, start with App Fundamentals.

To begin coding right away, read Building Your First App.

If you're new to Android development, it's important that you understand the following fundamental concepts about the Android app framework:

## Apps provide multiple entry points

Android apps are built as a combination of distinct components that can be invoked individually. For instance, an individual *activity* provides a single screen for a user interface, and a *service* independently performs work in the background.

From one component you can start another component using an *intent*. You can even start a component in a different app, such as an activity in a maps app to show an address. This model provides multiple entry points for a single app and allows any app to behave as a user's "default" for an action that other apps may invoke.

## Apps adapt to different devices

Android provides an adaptive app framework that allows you to provide unique resources for different device configurations. For example, you can create different XML layout files for different screen sizes and the system determines which layout to apply based on the current device's screen size.

You can query the availability of device features at runtime if any app features require specific hardware such as a camera. If necessary, you can also declare features your app requires so app markets such as Google Play Store do not allow installation on devices that do not support that feature.

**Learn more:**

# App Components

Android's application framework lets you create rich and innovative apps using a set of reusable components. This section explains how you can build the components that define the building blocks of your app and how to connect them together using intents.

INTENTS AND INTENT FILTERS >

BLOG ARTICLES

### Using DialogFragments

In this post, I'll show how to use DialogFragments with the v4 support library (for backward compatibility on pre-Honeycomb devices) to show a simple edit dialog and return a result to the calling Activity using an interface.

### Fragments For All

Today we've released a static library that exposes the

TRAINING

### Managing the Activity Lifecycle

This class explains important lifecycle callback methods that each Activity instance receives and how you can use them so your activity does what the user expects and does not consume system resources when your activity doesn't need them.

### Building a Dynamic UI with Fragments

This class shows you how to create a dynamic user

# User Interface

Your app's user interface is everything that the user can see and interact with. Android provides a variety of pre-built UI components such as structured layout objects and UI controls that allow you to build the graphical user interface for your app. Android also provides other UI modules for special interfaces such as dialogs, notifications, and menus.

OVERVIEW >

BLOG ARTICLES

### Say Goodbye to the Menu Button

As Ice Cream Sandwich rolls out to more devices, it's important that you begin to migrate your designs to the action bar in order to promote a consistent Android user experience.

### New Layout Widgets: Space and GridLayout

Ice Cream Sandwich (ICS) sports two new widgets d to support the richer user

TRAINING

### Implementing Effective Navigation

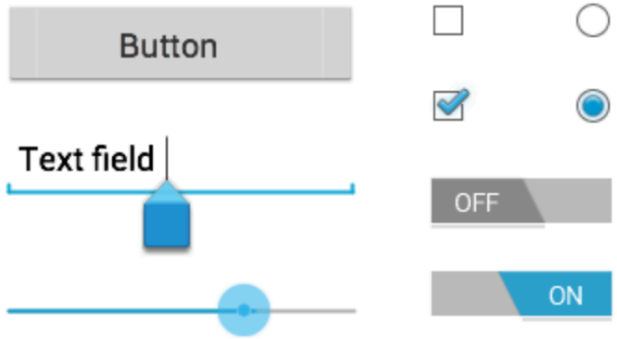
This class shows you how to plan out the high-level screen hierarchy for your application and then choose appropriate forms of navigation to allow users to effectively and intuitively traverse your content.

### Designing for Multiple Screens

Android powers hundreds of device types with several different screen sizes, ranging from small

# Input Controls

Input controls are the interactive components in your app's user interface. Android provides a wide variety of controls you can use in your UI, such as buttons, text fields, seek bars, checkboxes, zoom buttons, toggle buttons, and many more.




Adding an input control to your UI is as simple as adding an XML element to your XML layout. For example, here's a layout with a text field and button:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal">
    <EditText android:id="@+id/edit_message"
        android:layout_weight="1"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:hint="@string/edit_message" />
    <Button android:id="@+id/button_send"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button_send"
        android:onClick="sendMessage" />
</LinearLayout>
```

# Common Controls

| Control Type | Description | Related Classes |
|---|---|---|
| Button | A push-button that can be pressed, or clicked, by the user to perform an action. | `Button` |
| Text field | An editable text field. You can use the `AutoCompleteTextView` widget to create a text entry widget that provides auto-complete suggestions | `EditText` , `AutoCompleteTextView` |
| Checkbox | An on/off switch that can be toggled by the user. You should use checkboxes when presenting users with a group of selectable options that are not mutually exclusive. | `CheckBox` |
| Radio button | Similar to checkboxes, except that only one option can be selected in the group. | `RadioGroup` `RadioButton` |
| Toggle button | An on/off button with a light indicator. | `ToggleButton` |
| Spinner | A drop-down list that allows users to select one value from a set. | `Spinner` |
| Pickers | A dialog for users to select a single value for a set by using up/down buttons or via a swipe gesture. Use a `DatePicker` code> widget to enter the values for the date (month, day, year) or a `TimePicker` widget to enter the values for a time (hour, minute, AM/PM), which will be formatted automatically for the user's locale. | `DatePicker` , `TimePicker` |

# Buttons

A button consists of text or an icon (or both text and an icon) that communicates what action occurs when the user touches it.


Depending on whether you want a button with text, an icon, or both, you can create the button in your layout in three ways:

- With text, using the `Button` class:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_text"
    ... />
```

- With an icon, using the `ImageButton` class:

```
<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/button_icon"
    ... />
```

In this document



Key classes

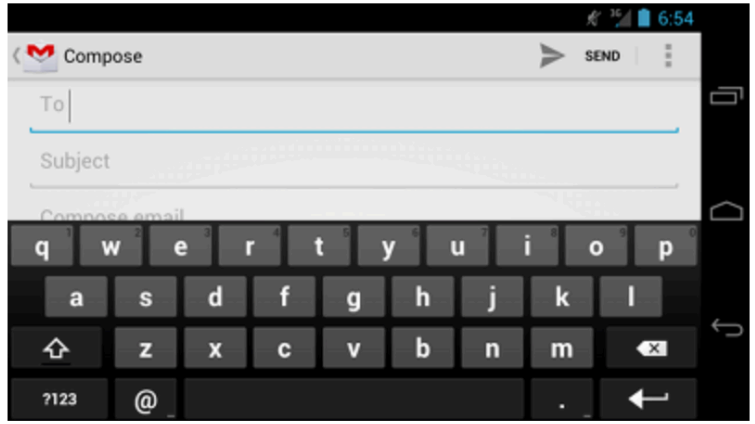
- `Button`
- `ImageButton`

Open “developer.android.com/guide/topics/ui/controls/button.html” in a new tab behind the current one `Button` class with the `android:drawableLeft` attribute:

# Text Fields

A text field allows the user to type text into your app. It can be either single line or multi-line. Touching a text field places the cursor and automatically displays the keyboard. In addition to typing, text fields allow for a variety of other activities, such as text selection (cut, copy, paste) and data look-up via auto-completion.

You can add a text field to you layout with the `EditText` object. You should usually do so in your XML layout with a `<EditText>` element.




In this document



Key classes

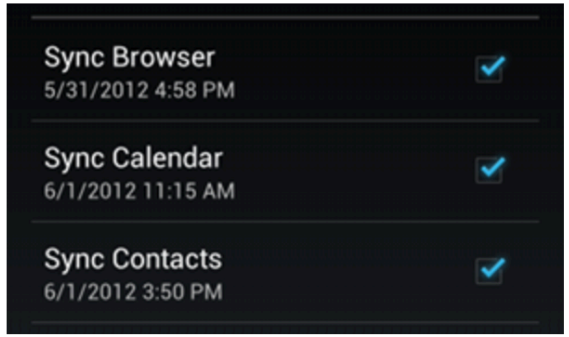
- `EditText`
- `AutoCompleteTextView`

## Specifying the Keyboard Type

ut types, such as number,

# Checkboxes

Checkboxes allow the user to select one or more options from a set. Typically, you should present each checkbox option in a vertical list.






Key classes
`CheckBox`

To create each checkbox option, create a `CheckBox` in your layout. Because a set of checkbox options allows the user to select multiple items, each checkbox is managed separately and you must register a click listener for each one.

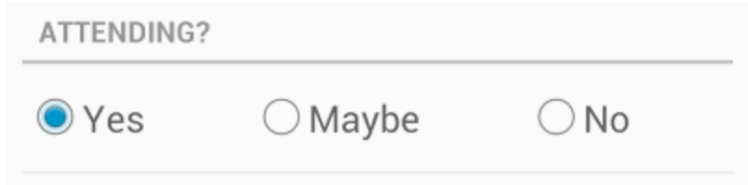
## Responding to Click Events

When the user selects a checkbox, the `CheckBox` object receives an on-click event.

To define the click event handler for a checkbox, add the `android:onClick` attribute to the `<CheckBox>` element in your XML layout. The value for this attribute must be the name of the method you want to call in response to a click event. The `Activity` hosting the layout must then implement the corresponding method.

# Radio Buttons

Radio buttons allow the user to select one option from a set. You should use radio buttons for optional sets that are mutually exclusive if you think that the user needs to see all available options side-by-side. If it's not necessary to show all options side-by-side, use a spinner instead.






To create each radio button option, create a `RadioButton` in your layout. However, because radio buttons are mutually exclusive, you must group them together inside a `RadioGroup`. By grouping them together, the system ensures that only one radio button can be selected at a time.

## Responding to Click Events

When the user selects one of the radio buttons, the corresponding `RadioButton` object receives an on-click event.

To define the click event handler for a button, add the `android:onClick` attribute to the `<RadioButton>` element in your XML layout. The value for this attribute must be the name of the method you want to call in response to a click event. The `Activity` hosting the layout must then implement the corresponding method.

# Toggle Buttons

A toggle button allows the user to change a setting between two states.

You can add a basic toggle button to your layout with the `ToggleButton` object. Android 4.0 (API level 14) introduces another kind of toggle button called a switch that provides a slider control, which you can add with a `Switch` object.

If you need to change a button's state yourself, you can use the `CompoundButton.setChecked()` or `CompoundButton.toggle()` methods.


*Toggle buttons*


*Switches (in Android 4.0+)*

In this document



Key classes

- `ToggleButton`
- `Switch`
- `CompoundButton`

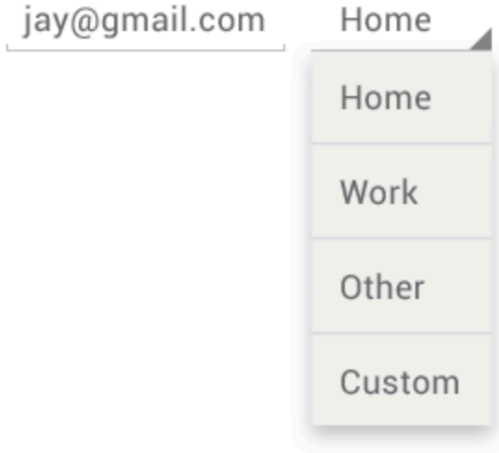
## Responding to Button Presses

To detect when the user activates the button or switch, create an `CompoundButton.OnCheckedChangeListener` object and assign it to the button by calling `setOnCheckedChangeListener()`. For example:

```
ToggleButton toggle = (ToggleButton) findViewById(R.id.togglebutton);
toggle.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
        if (isChecked) {
```

# Spinners

Spinners provide a quick way to select one value from a set. In the default state, a spinner shows its currently selected value. Touching the spinner displays a dropdown menu with all other available values, from which the user can select a new one.




In this document



Key classes

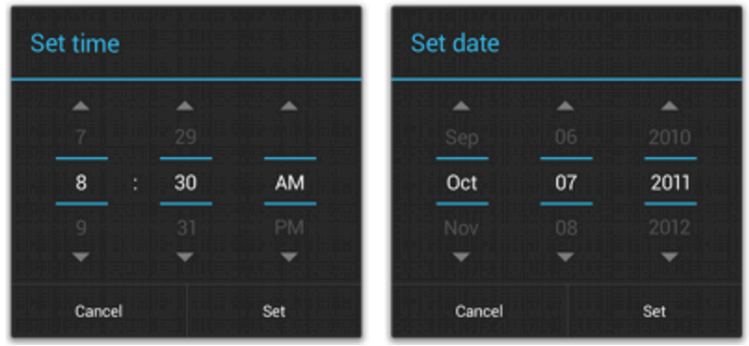
- `Spinner`
- `SpinnerAdapter`
- `AdapterView.OnItemSelectedListener`

You can add a spinner to your layout with the `Spinner` object. You should usually do so in your XML layout with a `<Spinner>` element. For example:

```
<Spinner
    android:id="@+id/planets_spinner"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" />
```

# Pickers

Android provides controls for the user to pick a time or pick a date as ready-to-use dialogs. Each picker provides controls for selecting each part of the time (hour, minute, AM/PM) or date (month, day, year). Using these pickers helps ensure that your users can pick a time or date that is valid, formatted correctly, and adjusted to the user's locale.

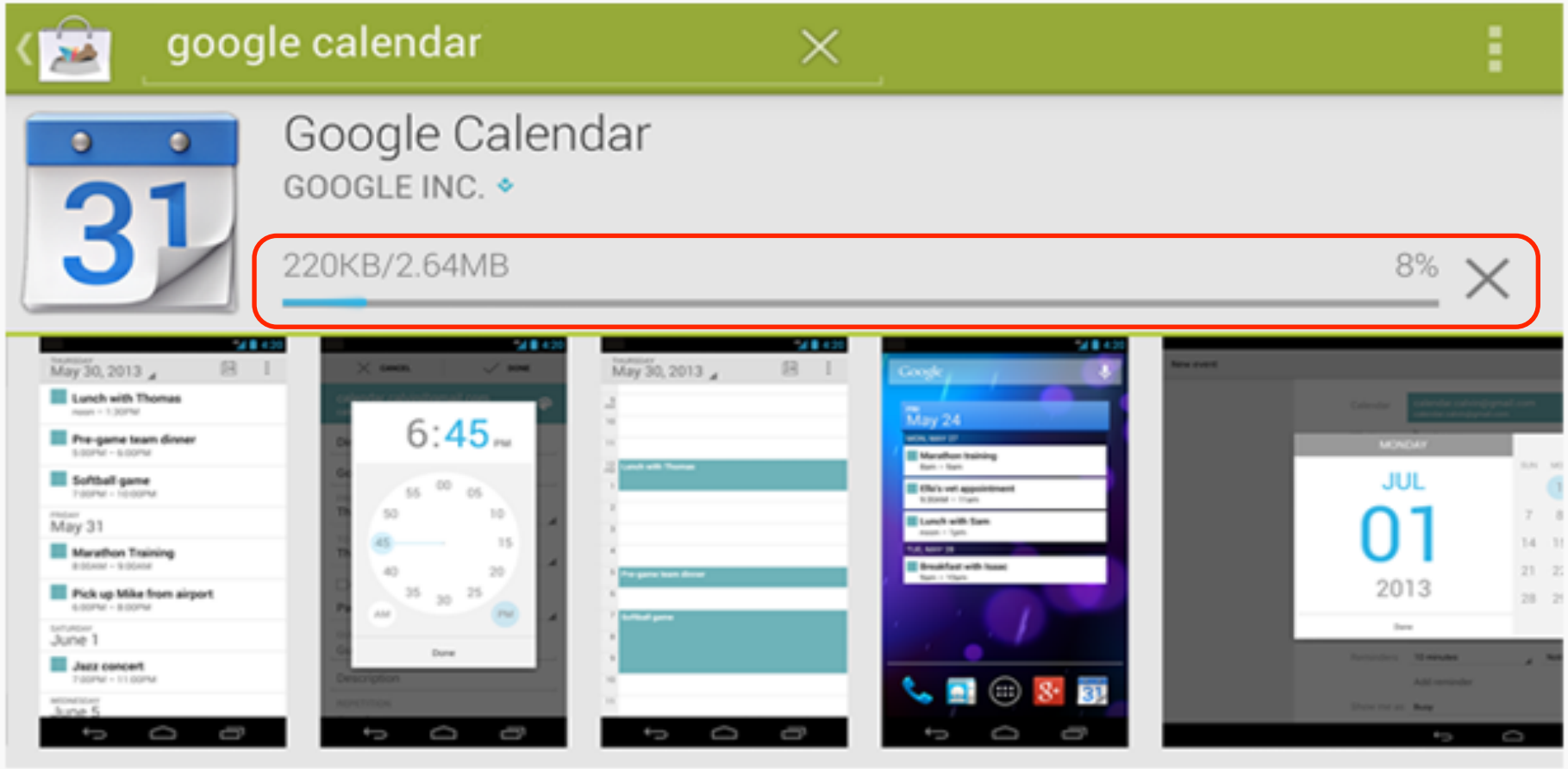



We recommend that you use `DialogFragment` to host each time or date picker. The `DialogFragment` manages the dialog lifecycle for you and allows you to display the pickers in different layout configurations, such as in a basic dialog on handsets or as an embedded part of the layout on large screens.

Although `DialogFragment` was first added to the platform in Android 3.0 (API level 11), if your app supports versions of Android older than 3.0—even as low as Android 1.6—you can use the `DialogFragment` class that's

# Progress bars

Progress bars are for situations where the percentage completed can be determined. They give users a quick sense of how much longer an operation will take.




A progress bar should always fill from 0% to 100% and never move backwards to a lower value. If multiple operations are happening in sequence, use the progress bar to represent the delay as a whole, so that when the bar reaches 100%, it doesn't return back to 0%.

# Google Maps Android API

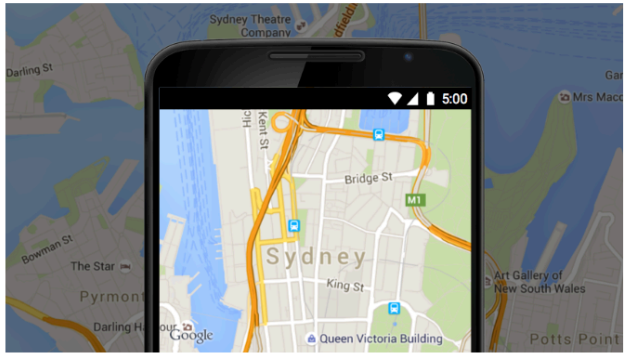
Add Google Maps to your Android app.

GET A KEY | VIEW PRICING AND PLANS

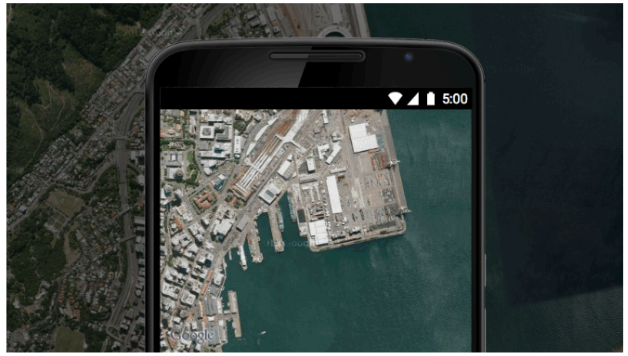


## The best of Google Maps for every Android app

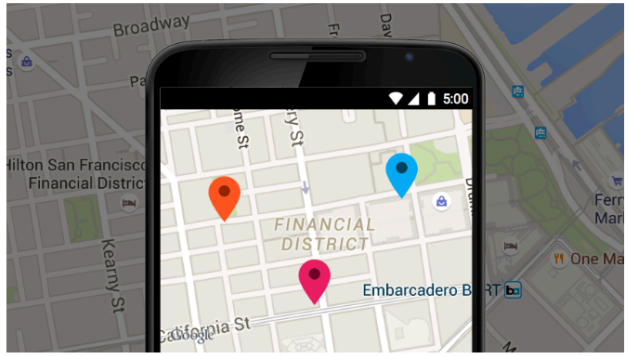
Build a custom map for your Android app using 3D buildings, indoor floor plans and more.




### Maps



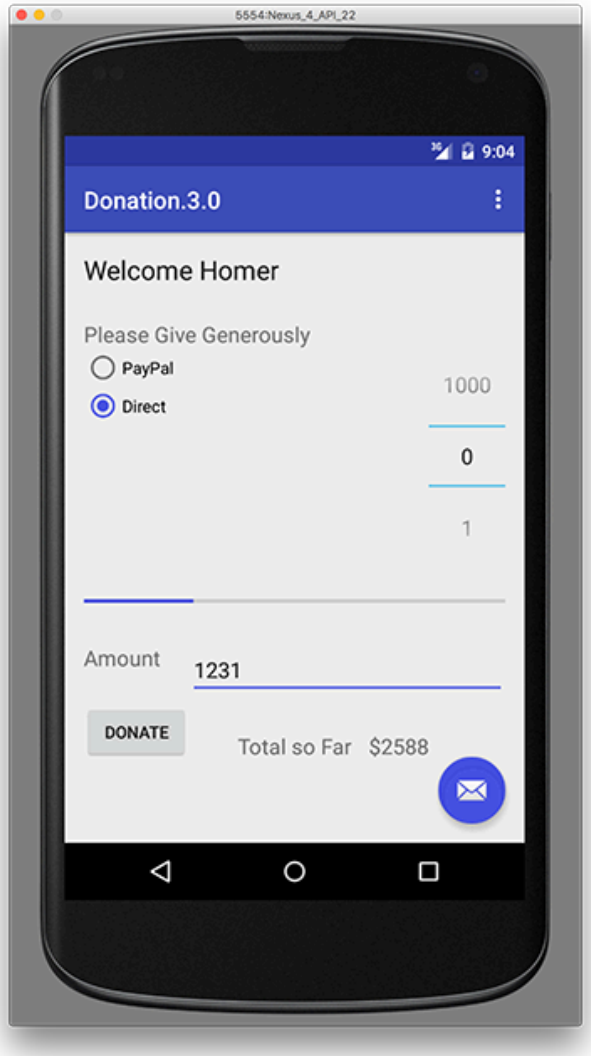

### Imagery




### Customization

# Case Study

- *Donation* – an Android App to keep track of donations made to '*Homers Presidential Campaign* '.
- App Features
  - Accept donation via number picker or typed amount
  - Keep a running total of donations
  - Display report on donation amounts and types
  - Display running total on progress bar



# Summary

- We looked at the Android Application Components
- Became aware of The Android Application Life Cycle
- Viewed the Online Developer Resources
- Took a very brief look at The "*Donation*" Case Study

# Questions?

# Appendix

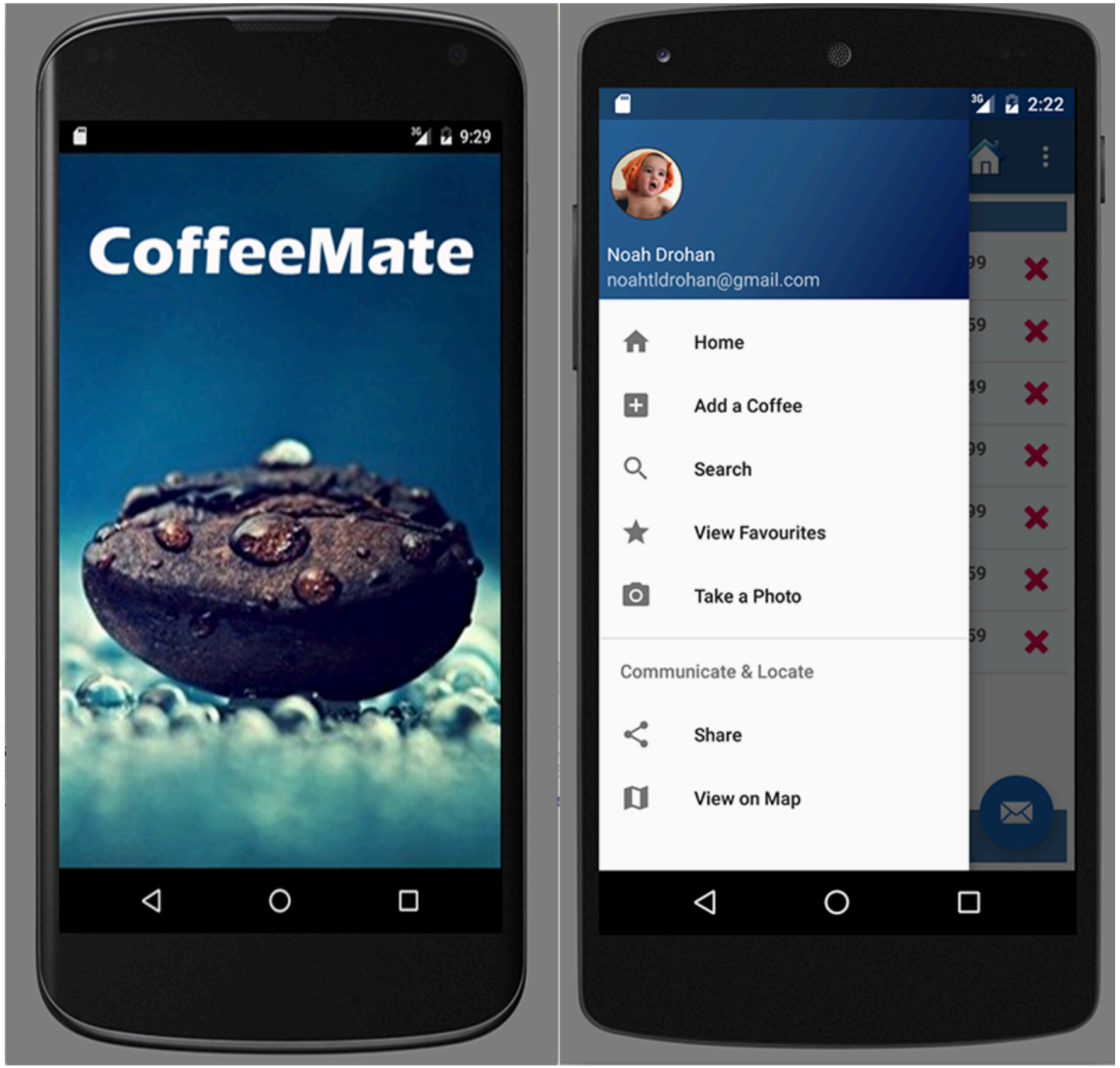
# Major Android Components

# Activities

- Activities manage (*control*) individual screens (*views*) with which a user could be interacting.
- Your program specifies a top level screen that runs upon application startup.
- Each Activity performs its own actions, to execute a method in, or launch, another Activity you use an *Intent*.
- The Activity base class already provides you with enough functionality to have a screen which “does nothing” - Provides you with an empty canvas…
- The activity allows you to set the top level GUI container.
- Then you instantiate some Views (widgets), put them in a container View (your layout), and set the container as the Activity’s top level View:
  - **`setContentView(View)`**
  - This is what gets displayed on the screen when the Activity is running.
  - We won’t go too in depth on GUI programming here, lots of documentation.
- The Activity is loaded by the Android OS, then the appropriate methods are called based on user interaction (back button?)

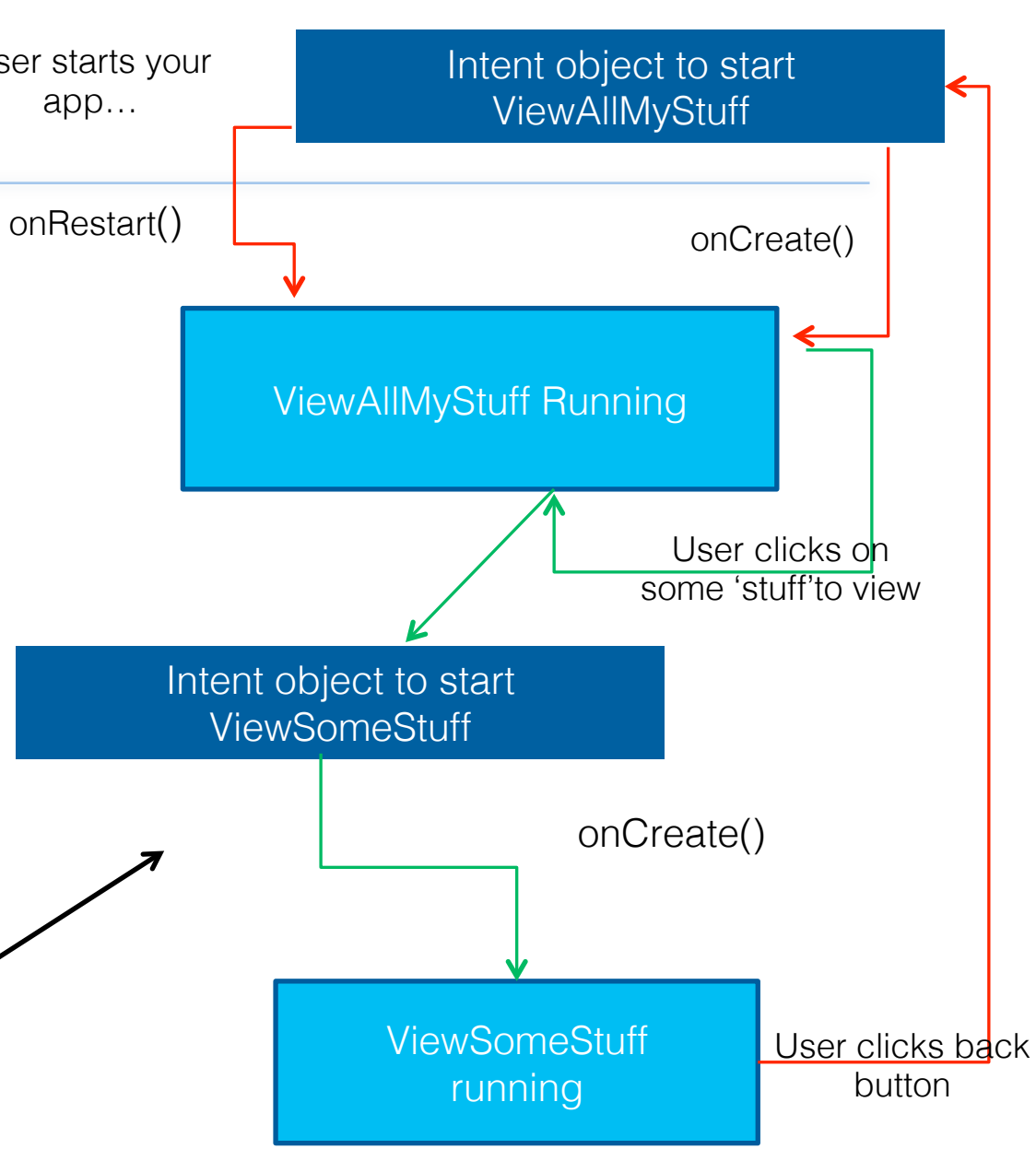
# Views

- The View class is the basic User Interface (UI) building block within Android and serves as the base class for nearly all the widgets and layouts within the SDK.
- The UI of an *Activity* (or a *Fragment* ) is built with widgets classes (Button, TextView, EditText, etc) which inherent from "`android.view.View`".
- Layout of the views is managed by "`android.view.ViewGroups`".



# Intents

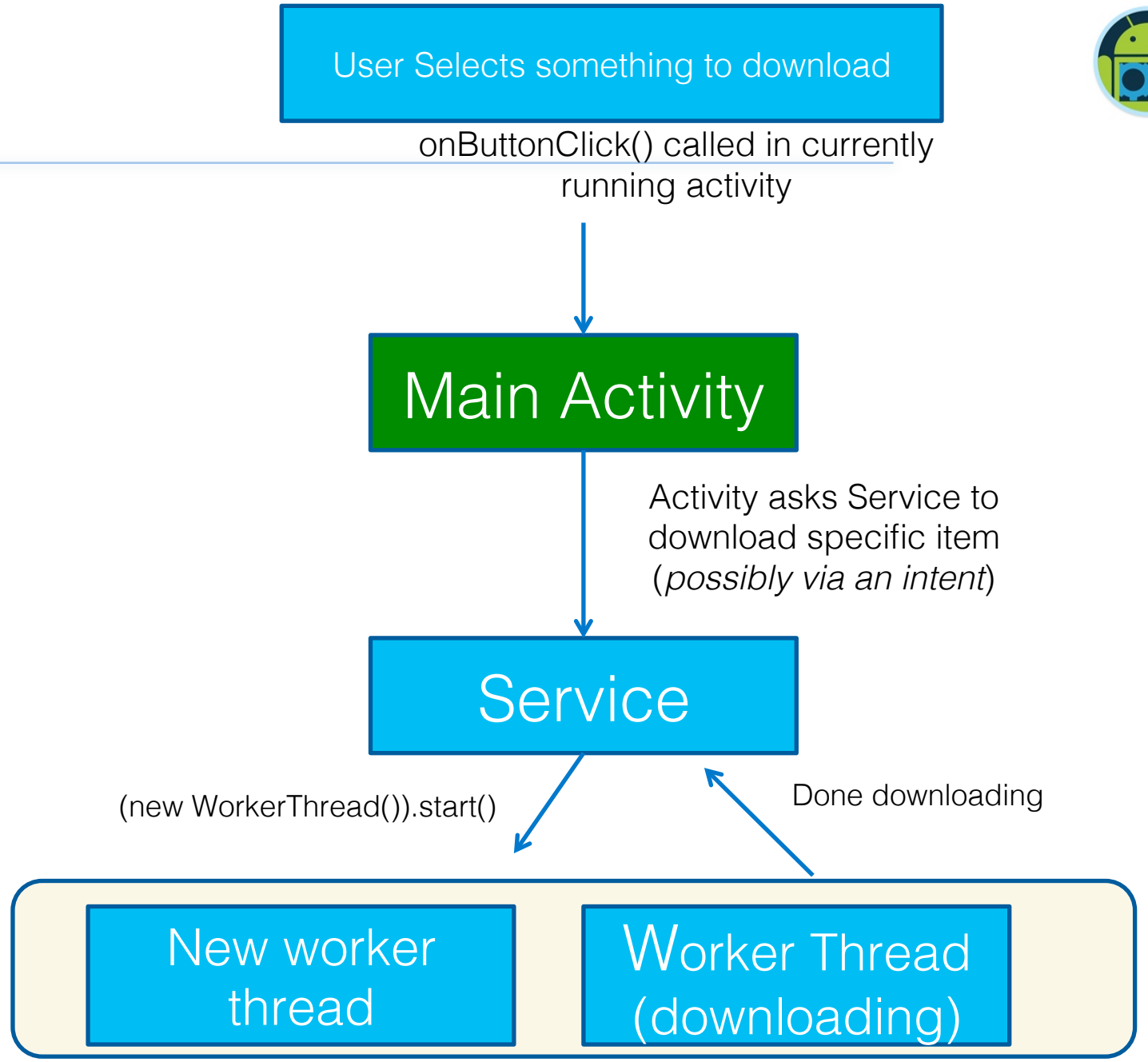
- How does an Activity (or any other runnable Android object) get started?
- We use the Intent class to ask the Android OS to start some Activity, Service, etc…
- Then the OS schedules that Activity to run, and that Activity has its onCreate (or onResume, etc…) method called.
- Intents are used to represent most inter-process requests in Android:
  - Dialing a number
  - Sending a text
  - Starting a new Activity within your application
- So the system will generate Intents, and so will your app!




*Notice! Here the red transitions are the events initiated by the Android OS, and the green transitions are created by your application*

# Services

- Services provide a way for your application to handle events in the background, without being explicitly associated with a View.
- However, services **don't** reside in their own thread
  - So don't perform things like network connections in a service, you will block the main thread
- What can you do?
  - Use your Service class to provide an interface to a background thread
  - Can call back to main activity using a Handler class
  - **`AsyncTask`** class



# Content Providers

- A component that stores and retrieves data and make it accessible to all applications.
  - uses a standard interface (URI) to fulfill requests for data from other applications & it's the only way to share data across applications.
    - `android.provider.Contacts.Phones.CONTENT_URI`
  - Android ships with a number of content providers for common data types (audio, video, images, personal contact information, and so on) - SQLite DB
  - Android 4.0 introduces the Calendar Provider.
    - `Calendars.CONTENT_URI;`

# SQLite - Persistence

- Eventually you'll want to be able to store data beyond the lifetime of your app.
  - You can use the SharedPreferences class to store simple key-value pairs
  - Simple interface, call getSharedPreferences and then use call getString, getBoolean, etc…
- However, you'll probably want to use more complicated storage.
  - Android provides a Bundle class to share complex objects
  - And ContentProviders provide inter process data storage
- The best solution is to use the Android interface to SQLite:
  - Lightweight database based on SQL
  - Fairly powerful, can't notice the difference between SQLite and SQL unless you have a large database
- You make queries to the database in standard SQL:
  - "SELECT ID, CITY, STATE FROM STATION WHERE LAT_N > 51.7;"
- Then your application provides a handler to interface the SQL database to other applications via a content provider:

# Broadcast Receivers

- A component designed to respond to broadcast Intents.
  - Receives system wide messages and implicit intents
  - can be used to react to changed conditions in the system (external notifications or alarms).
  - An application can register as a broadcast receiver for certain events and can be started if such an event occurs. These events can come from Android itself (e.g., battery low) or from any program running on the system.
- An Activity or Service provides other applications with access to its functionality by executing an Intent Receiver, a small piece of code that responds to requests for data or services from other activities.

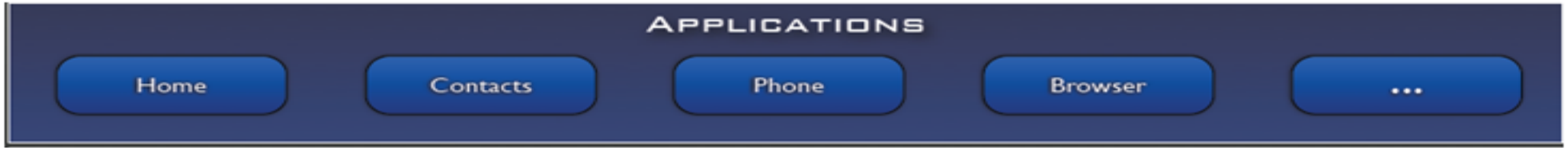
# The Layered Framework

slides paraphrase a blog post by Tim Bray (co-inventor of XML and currently employed by Google to work on Android)

http://www.tbray.org/ongoing/When/201x/2010/11/14/What-Android-Is

# The Layered Framework (1)

## ❑ Applications Layer




- Android provides a set of core applications:
  - ✓ Email Client
  - ✓ SMS Program
  - ✓ Calendar
  - ✓ Maps
  - ✓ Browser
  - ✓ Contacts
  - ✓ YOUR APP
  - ✓ Etc
- All applications are written using the Java language. These applications are executed by the Dalvik Virtual Machine (DVM), similar to a Java Virtual Machine but with different bytecodes♪

# The Layered Framework (2)

## Application Framework Layer


- Enabling and simplifying the reuse of components
  - Developers have full access to the same framework APIs used by the core applications.
  - Users are allowed to replace components.
- These services are used by developers to create Android applications that can be run in the emulator or on a device

- See next slide for more…..

# The Layered Framework (3)

## ❑ Application Framework Layer Features

| Feature♪ | Role♪ |
|---|---|
| View System♪ | Used to build an application, including lists, grids, text boxes, buttons, and embedded web browser♪ |
| Content Provider♪ | Enabling applications to access data from other applications or to share their own data♪ |
| Resource Manager♪ | Providing access to non-code resources (localized strings, graphics, and layout files)♪ |
| Notification Manager♪ | Enabling all applications to display custom alerts in the status bar♪ |
| Activity Manager♪ | Managing the lifecycle of applications and providing a common navigation (back) stack♪ |

We'll be covering the above in more detail later on...

# The Layered Framework (4)

## Libraries Layer


- Including a set of C/C++ libraries used by components of the Android system
- Exposed to developers through the Android application framework

**System C library/libc** - a BSD (Berkeley Software Distribution) -derived implementation of the standard C system library (libc), tuned for embedded Linux-based devices

**Media Framework/Libraries** - based on PacketVideo's OpenCORE; the libraries support playback and recording of many popular audio and video formats, as well as static image files, including MPEG4, H.264, MP3, AAC, AMR, JPG, and PNG

**Surface Manager** - manages access to the display subsystem and seamlessly composites 2D and 3D graphic layers from multiple applications

**WebKit/LibWebCore** - a modern web browser engine which powers both the Android browser and an embeddable web view

**SGL ( Scene Graph Library)** - the underlying 2D graphics engine

**3D libraries** - an implementation based on **OpenGL ES** 1.0 APIs; the libraries use either hardware 3D acceleration (where available) or the included, highly optimized 3D software rasterizer (shapes->pixels)

**FreeType** - bitmap and vector font rendering

**SQLite** - a powerful and lightweight relational database engine available to all applications

# The Layered Framework (5)♪

## Core Runtime Libraries
(changing to ART in Kit Kat)


Next Slide

- Providing most of the functionality available in the core libraries of the Java language
- APIs
  - Data Structures
  - Utilities
  - File Access
  - Network Access
  - Graphics
  - Etc

# The Layered Framework (6)♪

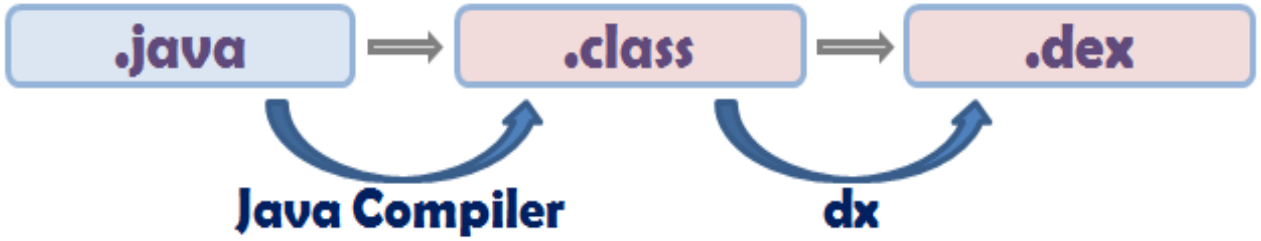
- Dalvik Virtual Machine (DVM)
  - Provides an environment on which every Android application runs
    - Each Android application runs in its own process, with its own instance of the Dalvik VM.
    - Dalvik has been written such that a device can run multiple VMs efficiently.
- Android Runtime (ART) 4.4

(see slide 12)

# The Layered Framework (7)♪

## ❑ Dalvik Virtual Machine (Cont'd)

- ✓ Executing the Dalvik Executable (.dex) format
  - ➢ .dex format is optimized for minimal memory footprint.
  - ➢ Compilation




- ✓ Relying on the Linux Kernel for:
  - ➢ Threading
  - ➢ Low-level memory management

♪

# ART – Android Runtime

- Handles app execution in a fundamentally different way from Dalvik.
- Current runtime relies on a JIT compiler to interpret original bytecode
  - In a manner of speaking, apps are only partially compiled by developers
  - resulting code must go through an interpreter on a user's device each and every time it is run == Overhead + Inefficient
  - But the mechanism makes it easy for apps to run on a variety of hardware and architectures.
- ART pre-compiles that bytecode into machine language when *apps are first installed*, turning them into truly native apps.
  - This process is called Ahead-Of-Time (AOT) compilation.
- By removing the need to spin up a new VM or run interpreted code, startup times can be cut down immensely and ongoing execution will become faster.

# The Layered Framework (8)

## Linux Kernel Layer


- At the bottom is the Linux kernel that has been augmented with extensions for Android
  - the extensions deal with power-savings, essentially adapting the Linux kernel to run on mobile devices
- Relying on Linux Kernel 2.6 for core system services / 3.8 in Kit Kat
  - Memory and Process Management
  - Network Stack
  - Driver Model
  - Security
- Providing an abstraction layer between the H/W and the rest of the S/W stack

# The Application/Activity Lifecycle

# The Activity Life Cycle

An activity monitors and reacts to these events by instantiating methods that override the Activity class methods for each event:

## ❑onCreate

- Called when an activity is first created. This is the place you normally create your views, open any persistent data files your activity needs to use, and in general initialize your activity.
- When calling onCreate(), the Android framework is passed a Bundle object that contains any activity state saved from when the activity ran before.

# The Activity Life Cycle

## ❑onStart

- Called just before an activity becomes visible on the screen. Once onStart() completes, if your activity can become the foreground activity on the screen, control will transfer to onResume().
- If the activity cannot become the foreground activity for some reason, control transfers to the onStop() method.

# The Activity Life Cycle

## ❑ onResume

- Called right after onStart() if your activity is the foreground activity on the screen. At this point your activity is running and interacting with the user. You are receiving keyboard and touch inputs, and the screen is displaying your user interface.
- onResume() is also called if your activity loses the foreground to another activity, and that activity eventually exits, popping your activity back to the foreground. This is where your activity would start (or resume) doing things that are needed to update the user interface.

# The Activity Life Cycle

## ❑onPause

- Called when Android is just about to resume a different activity, giving that activity the foreground. At this point your activity will no longer have access to the screen, so you should stop doing things that consume battery and CPU cycles unnecessarily.
  - If you are running an animation, no one is going to be able to see it, so you might as well suspend it until you get the screen back. Your activity needs to take advantage of this method to store any state that you will need in case your activity gains the foreground again—and it is not guaranteed that your activity will resume.
- Once you exit this method, Android may kill your activity at any time without returning control to you.

# The Activity Life Cycle

## ❑ onStop

- Called when your activity is no longer visible, either because another activity has taken the foreground or because your activity is being destroyed.

## ❑ onDestroy

- The last chance for your activity to do any processing before it is destroyed. Normally you'd get to this point because the activity is done and the framework called its finish method. But as mentioned earlier, the method might be called because Android has decided it needs the resources your activity is consuming.